Help Center

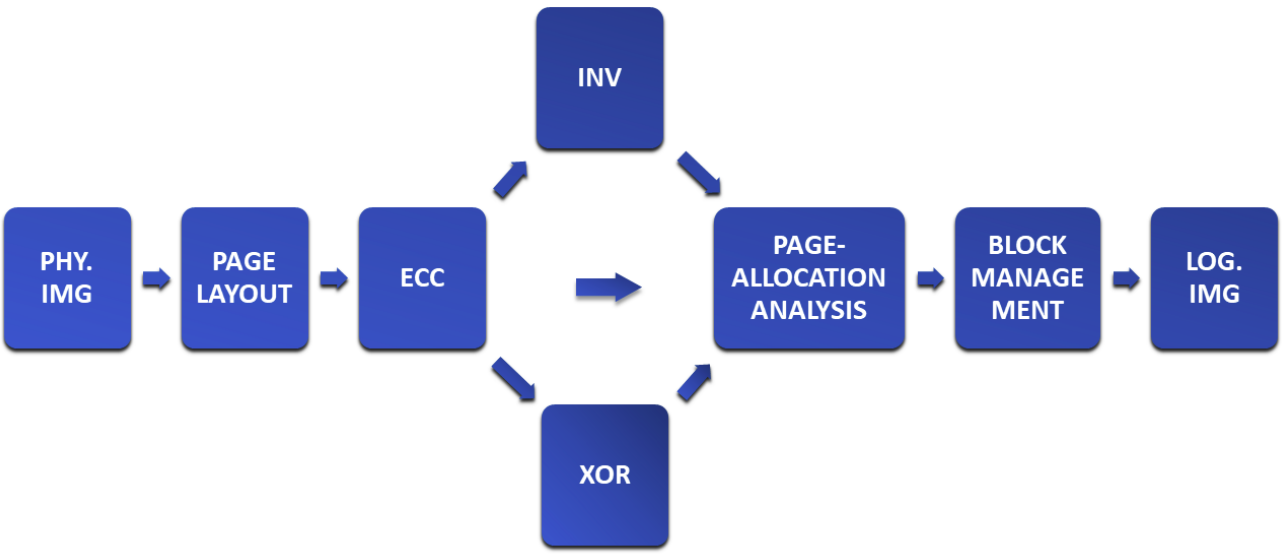
Silicon Motion(SM) controllers - Data recovery

We can find plenty of NAND controllers on the market and without any doubt one of the most popular is SiliconMotion. These controllers found their use in all kinds of flash devices, starting from USB flash drives, SD cards, monolithic devices, and ending on superfast CF cards or modern SSD drives.

In this article, we've collected all the required information you will need to process cases with SiliconMotion controllers on your own. You will find here the whole data recovery procedure and for each data reconstruction step, you will have a set of peculiarities and methods which will help you with physical image reconstruction.

When it comes to chip-off, recovery from these controllers isn't very complicated. In the majority of cases, the data reconstruction procedure will be looking exactly the same, since these controllers don't differ much between each model.

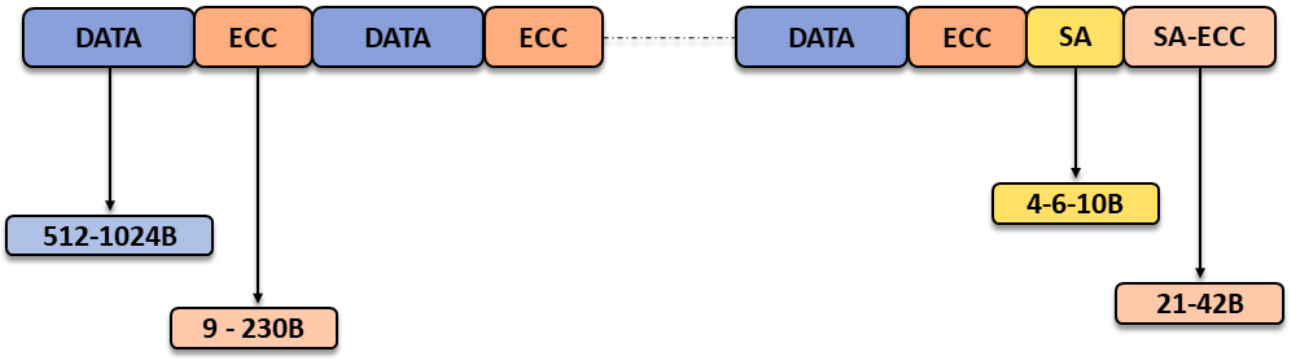
Chip-off data recovery, the procedure of physical image reconstruction.



> This article doesn't cover the very first step- chip reading, so if you are new, please take a look at this article: Chip Reading or the video: Chip reading. ECC usage. Page layout. Dump reread. Read Retry.

Page structure

SiliconMotion controllers have very characteristic page structures. The first part of a physical page consists of Data areas and ECC areas after which we have separated Service area with its own ECC checksum.



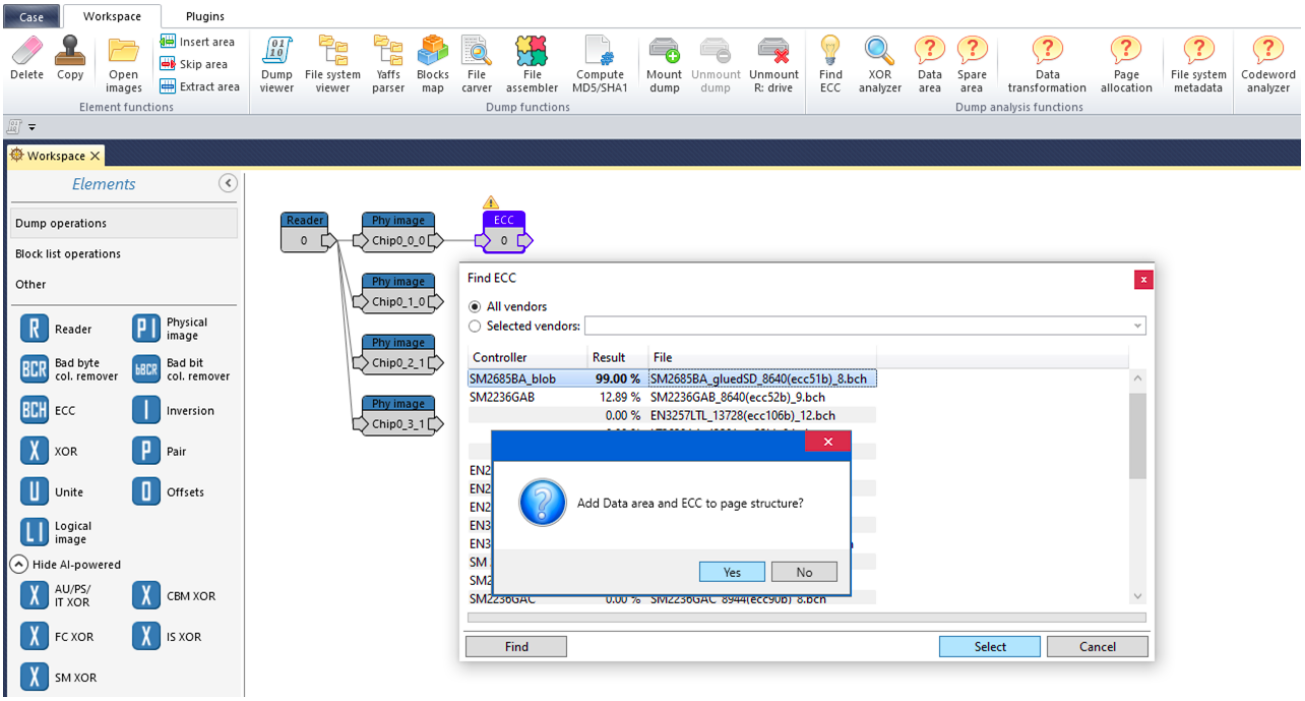
- Data area - SiliconMotion controllers may reserve either 512 or 1024 Bytes for data area structure.
- ECC area - The size of this area is configurable and may differ from case to case, even despite the same controller model. Sizes of checksums generated by the BCH algorithm are usually in a range from 9 up to 240 Bytes.
- Service area - This area may also have a few different sizes, usually, it has exactly 10B, however, you may also encounter 4B and 6B reserved for the Service area.
- ECC for Service area - These controllers have separated Service area with its own ECC checksum and the size of this area, in the majority of cases, is equal to 21/22B.

Determination of page structure

There are two methods to find and set the page structure for the cases with these controllers. You can try to do this automatically or set the page structure manually.

Automatic determination

After the ECC has been found, VNR will ask to set the page structure based on the selected BCH file. Then after committing, the page structure will be set.

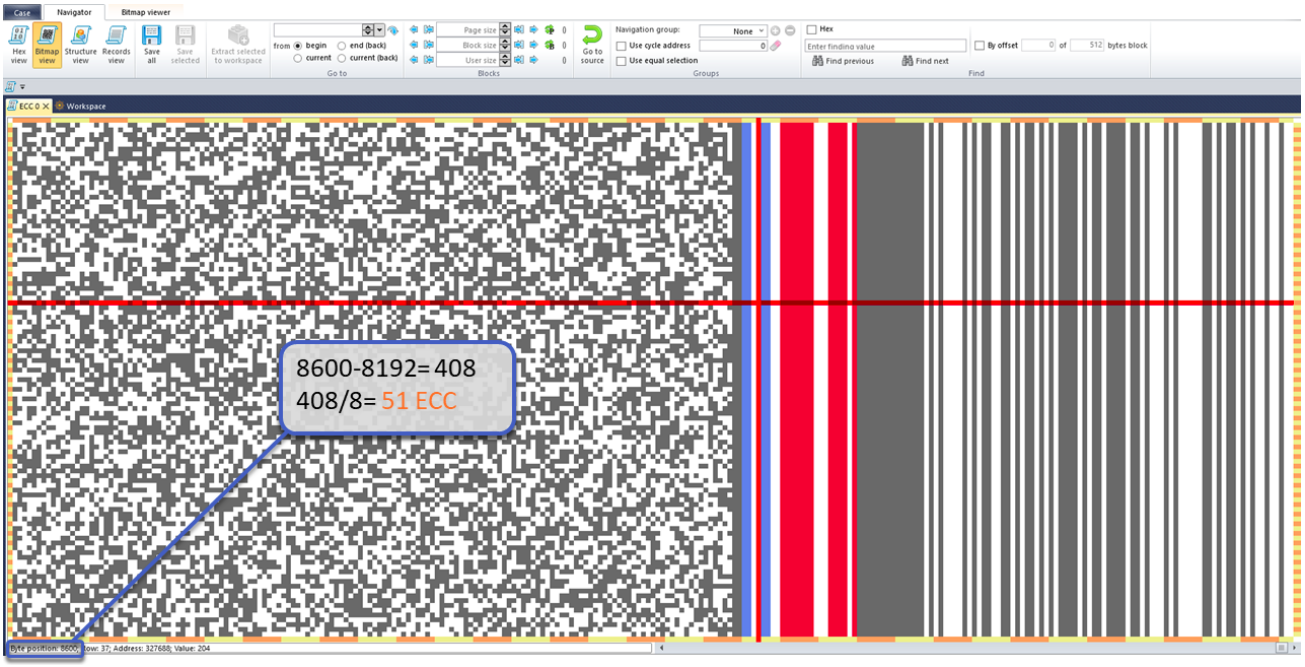


Manual page structure

In cases where ECC hasn't been found, it's necessary to set the page structure manually. Manual determination of the page structure may be a little bit tickly due to the unknown size of ECC, which in the case of these controllers may have a lot of different sizes. Below you will find two approaches to determine the page structure manually.

The first method takes advantage of the specific page structure of these controllers.

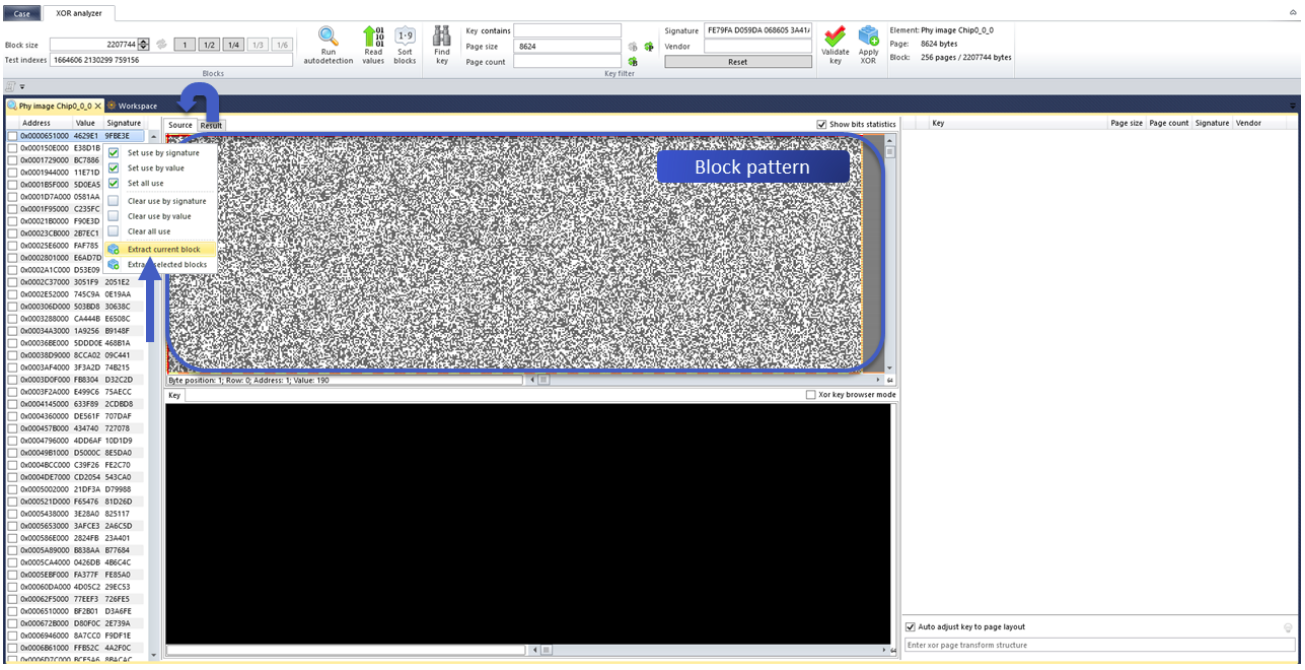
It's necessary to find and mark the very first byte of the Service area(**Header**) and check its position. Afterward, it's necessary to subtract from this value the total size of Data areas and divide the achieved value by the number of Data areas + ECC in the page. As result, we will get the size of the ECC area.



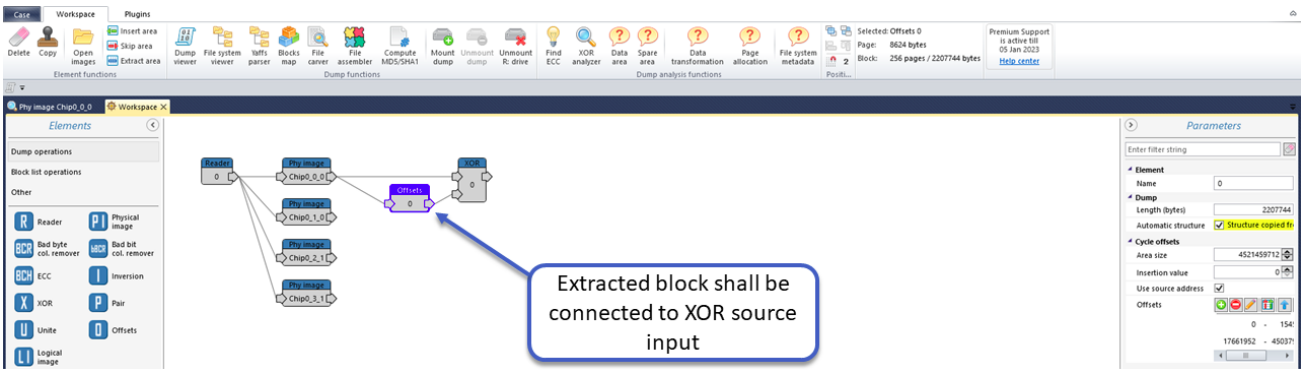
The second method finds its use in complicated cases when everything including SA is scrambled and XOR couldn't be detected. The approach in this method also isn't complex. Just a few operations will have to be performed.

Run XOR analyzer on physical image and wait till autodetection process will pass. Next extract one block with data(noise pattern) from the block menu, on the left side.

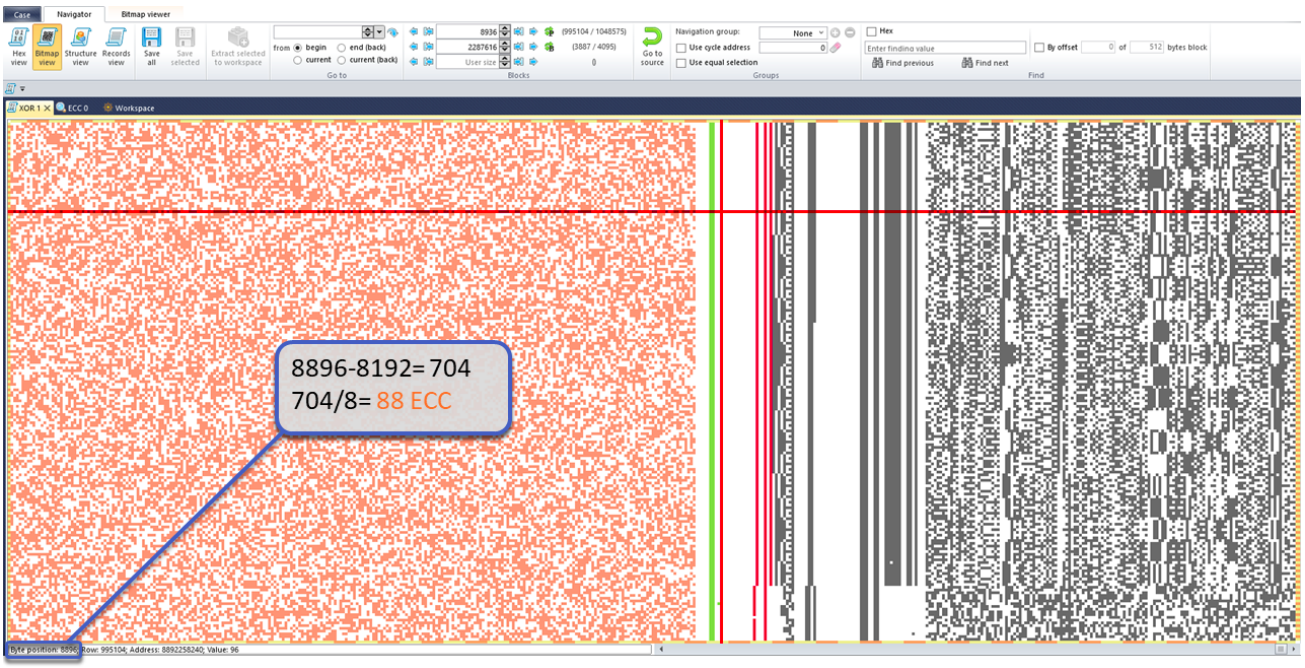
ⓘ Keep in mind that after autodetection XOR analyzer will automatically switch the view to the Result tab, in order to check the pattern of the selected block it's necessary to switch the view to the Source tab.



The extracted block needs to be connected as the XOR key with the physical image by using the XOR element.

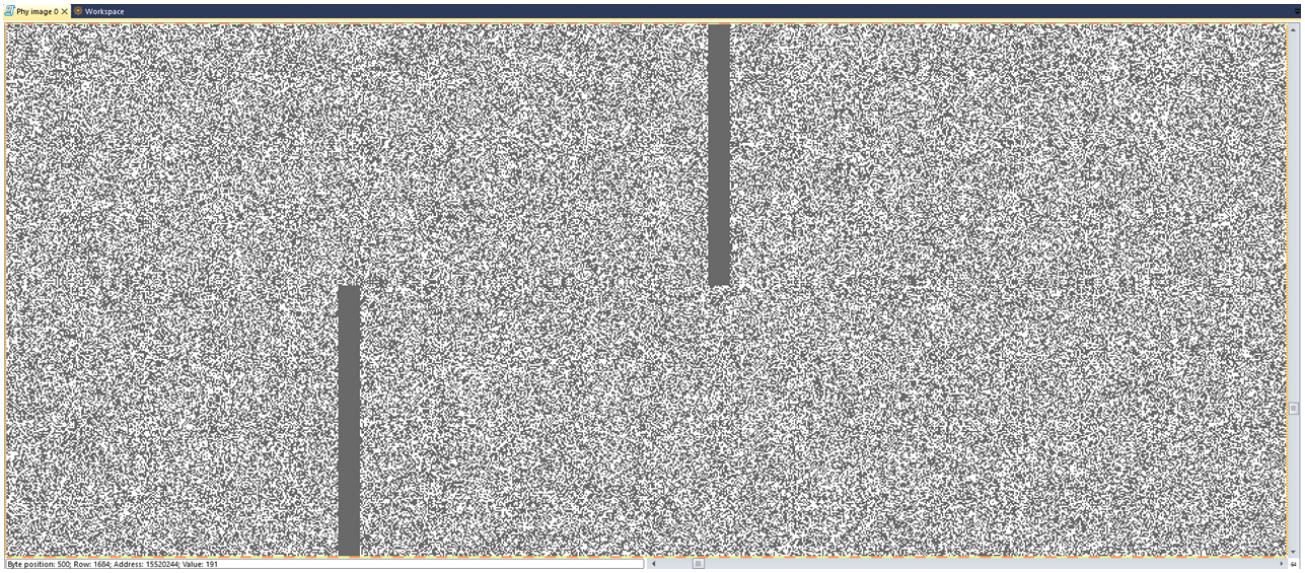


This operation will allow you to uncover the Service area and calculate page structure in the same way as for the first method.



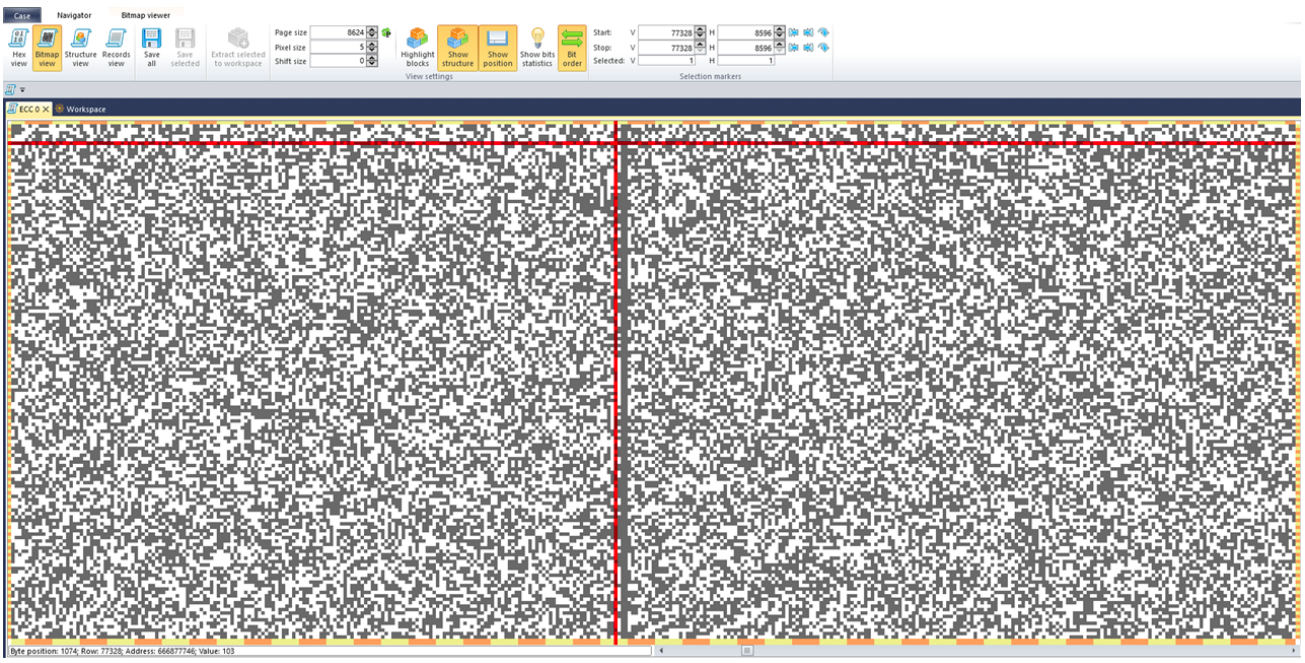
Bad columns

Bad columns in Silicon Motion controllers aren't complicated to notice and remove. Their pattern is always static and can be easily detected either manually or by using Bad Column Remover(BCR) element with preset for SM controllers.

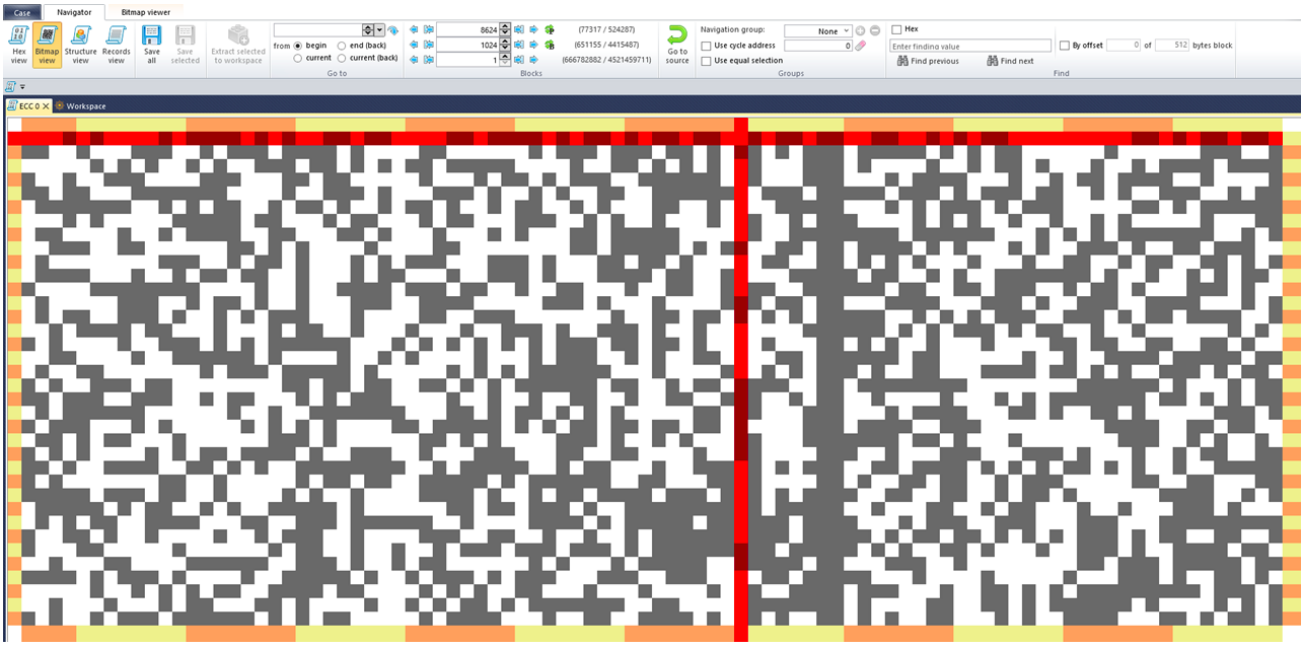


In many cases with SM controllers the generated ECC checksums aren't divisible by 8 bits, therefore controller is filling out the end of ECC with a couple of ones (bit 1), to fulfill the last byte. This can be easily spotted via bitmap viewer and in some cases, it may be confused with a real bad column.

> ⓘ There are also cases where ECC ends with even 15 ones, so one full Byte and 7 Bits. Keep that in mind during the determination of page layout.

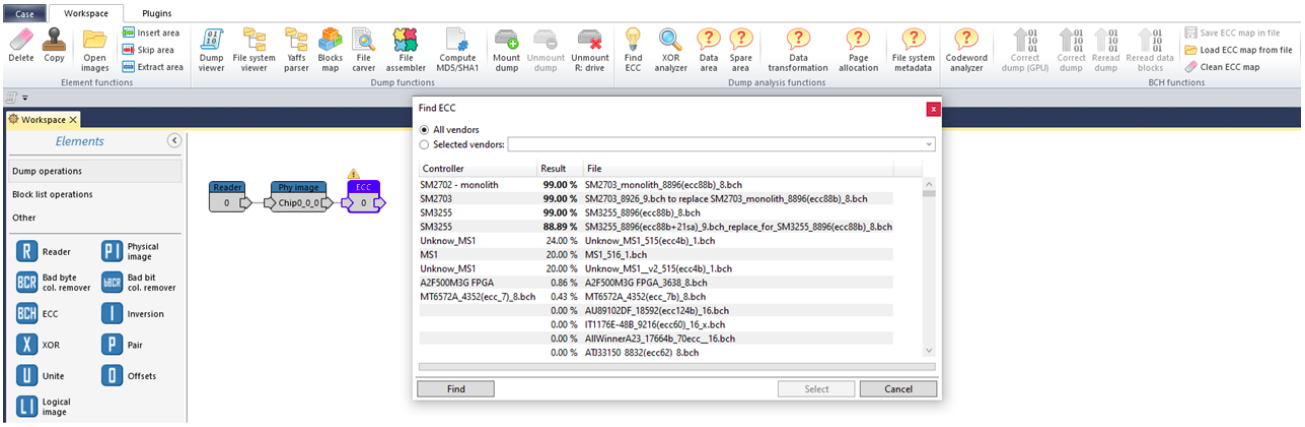


If we will take a look closer at this "column" we will be able to see that it has a few bits of ECC checksum and afterward we have only ones. It also allows to distinguish the last byte of ECC from a bad column.

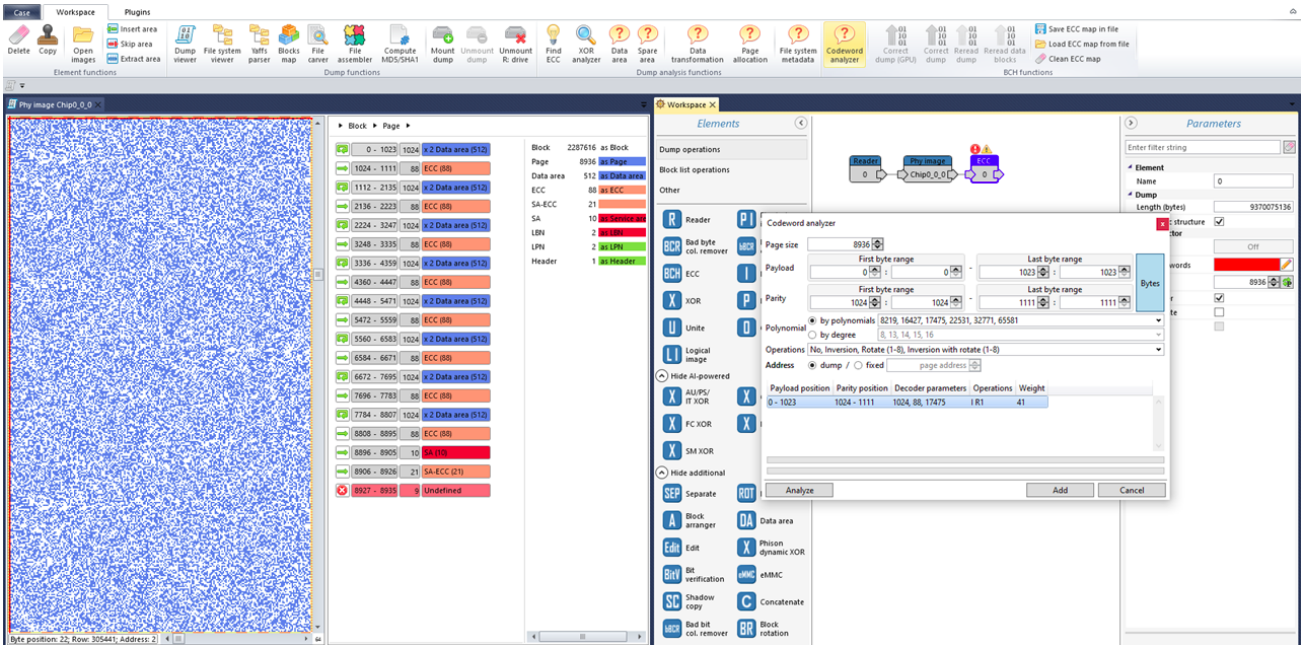


Error correction codes(ECC)

Like in every case, ECC can be found automatically with the Find ECC button on the top panel.



For the cases where the ECC couldn't be detected, it's necessary to use another tool. Codeword analyzer allows you to detect parameters of ECC manually and prepare the BCH file which can be placed later in a user VNR database.



ⓘ For more information about the usage of the Codeword analysis function, please take a look at this article - ECC in NAND flash memory, chapter Unsupported ECC - how to create a new code.

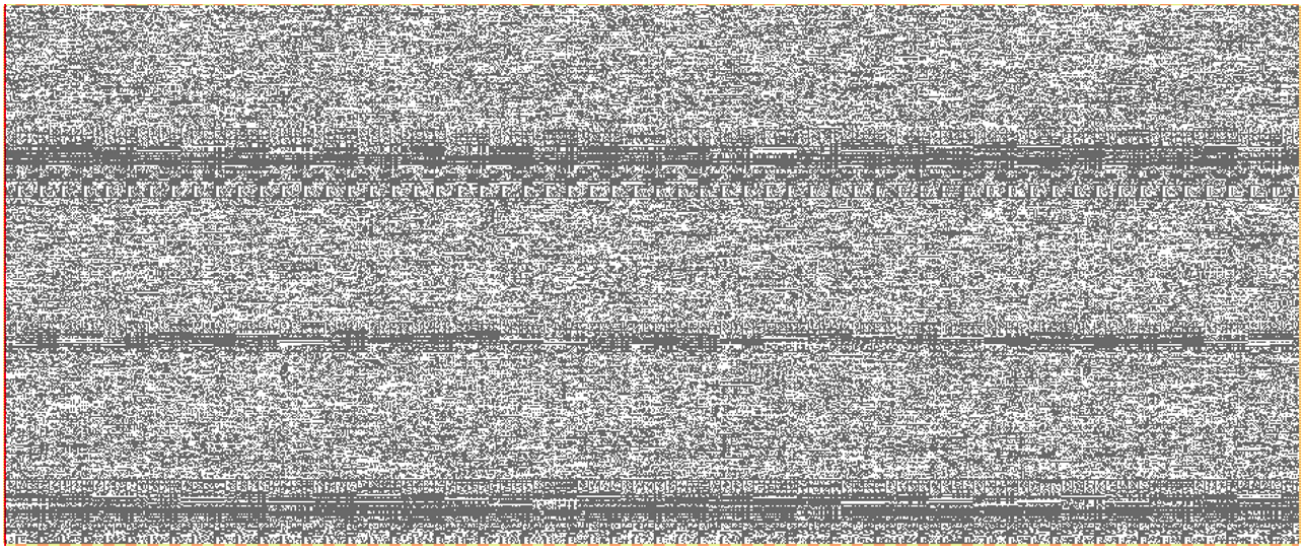
Data transformation

Transformations of the data streams that are being sent to NAND memory are part and parcel of every NAND controller designed to work with MLC/TLC chips. We can distinguish two types of transformation for SM controllers. Inversion and Scrambling. Each of these transformations can be checked by analysis of bitmap.

Inversion

Data transformation characteristics for SLC and early models of MLC chips. It can be found in old, low-capacity devices.

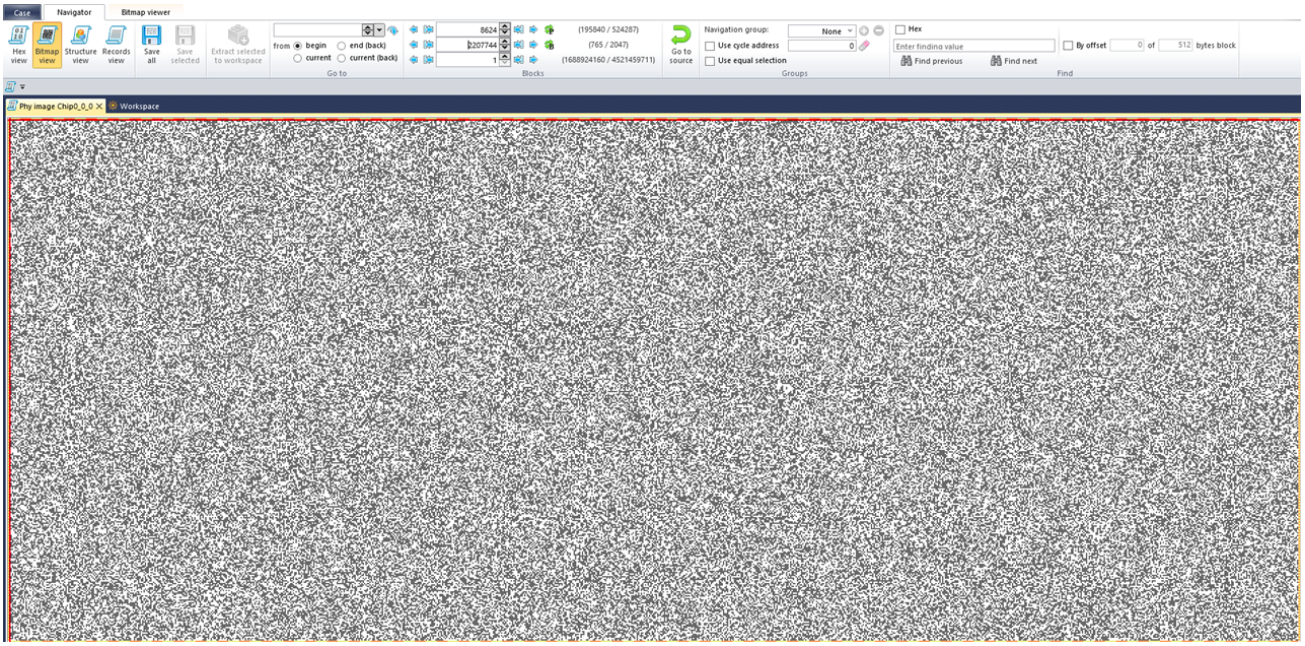
Binaries of user data, which were inverted, consist of ones (grey pixels) in the majority. In such cases, the Inversion element shall be used.



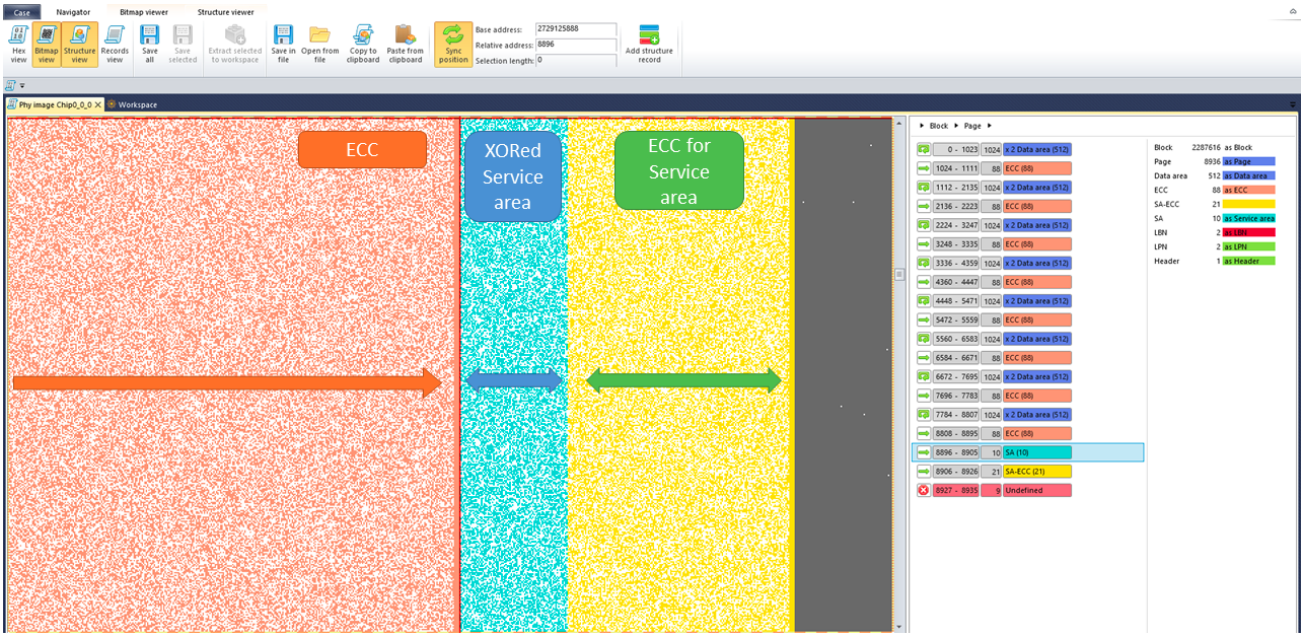
Scrambling (XOR key)

The most popular data transformation for modern flash devices. it's typical for High capacity devices equipped with TLC and some MLC chips.

Scrambled user data doesn't have any patterns. In such cases, only noise can be found in the Data areas via the Bitmap viewer of VNR.



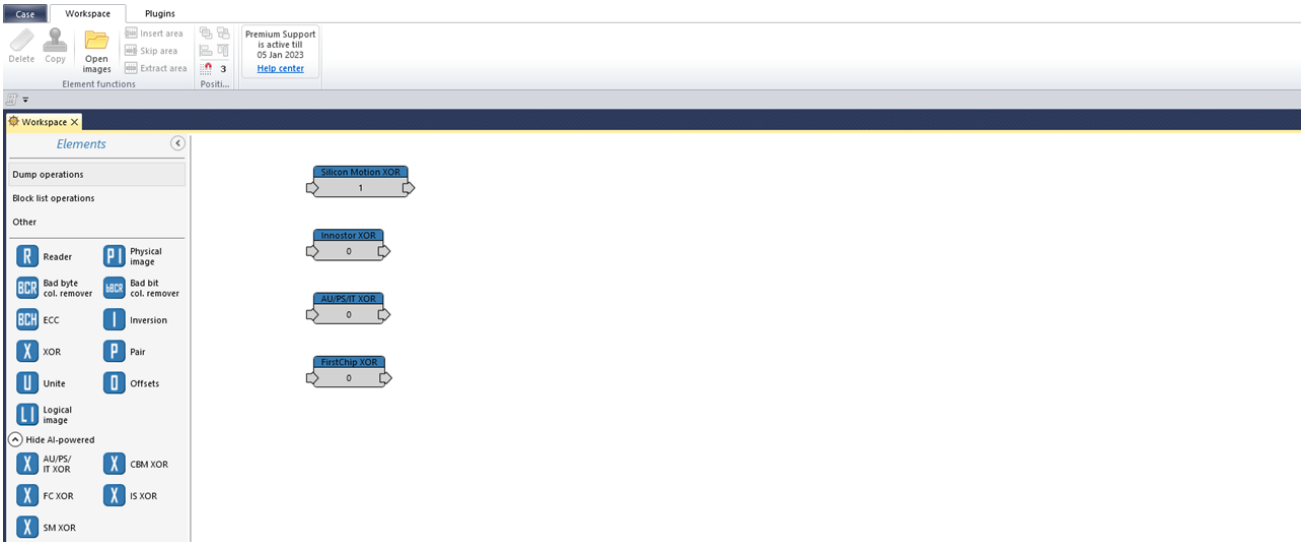
Controllers discussed in this article may apply the XOR key not only to the Data areas, but also to the Service area. In such cases, every structure via bit map viewer will be looking like noise.



SiliconMotion AI-powered XOR algorithm

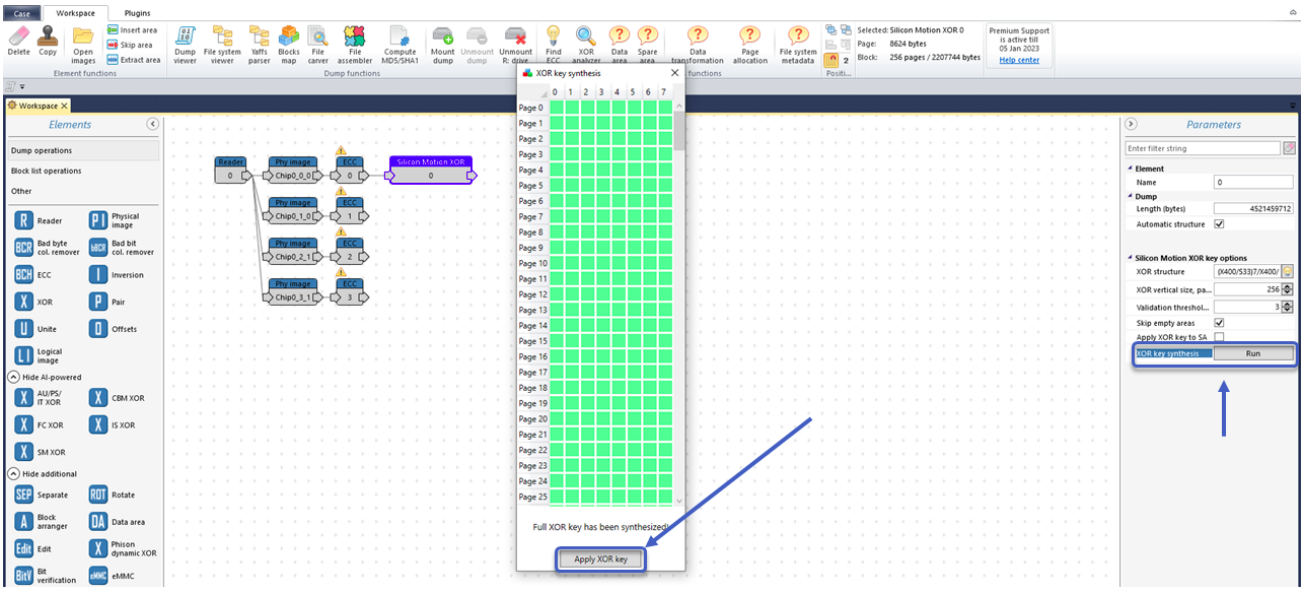
Each controller is designed to work with several different NAND memory chips which are usually physically different from each other. Capacity isn't the only thing that may differ, sizes of blocks and pages may be different as well. As a result, the XOR key which is generated by the controller may also be different, since it's generated for different NAND chips. Therefore there is a possibility that you will face a situation where the XOR key won't be available in the Database.

For such cases, VNR has a couple of elements that allow to synthesize the XOR key by analysis of the whole physical image. In VNR you can already find such elements for all the most popular brands, including SiliconMotion controllers, so in case when XOR isn't available in database this elements shall be used.

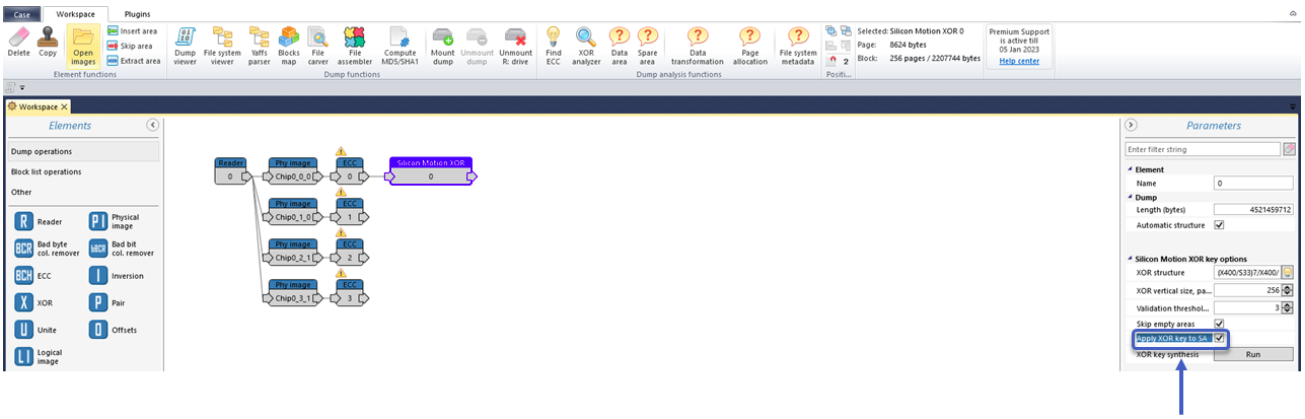


Each of these elements requires a full page structure to be set, then the

element can be connected and synthesis started. All the other parameters are being adjusted automatically and don't require to be checked.



By default these elements allow to prepare the XOR for the data area, however, the Silicon Motion XOR allows to generate the key for the Service area too. It requires the Service area structure to be set in Structure View.



These elements were presented and wider described on our update webinars. For more information about technology and usage of these elements please visit our YT channel, you will find there recordings from every webinar and much more instructional videos.

ⓘ **SM XOR** - VISUAL NAND RECONSTRUCTOR 5.0

Data Allocation modes

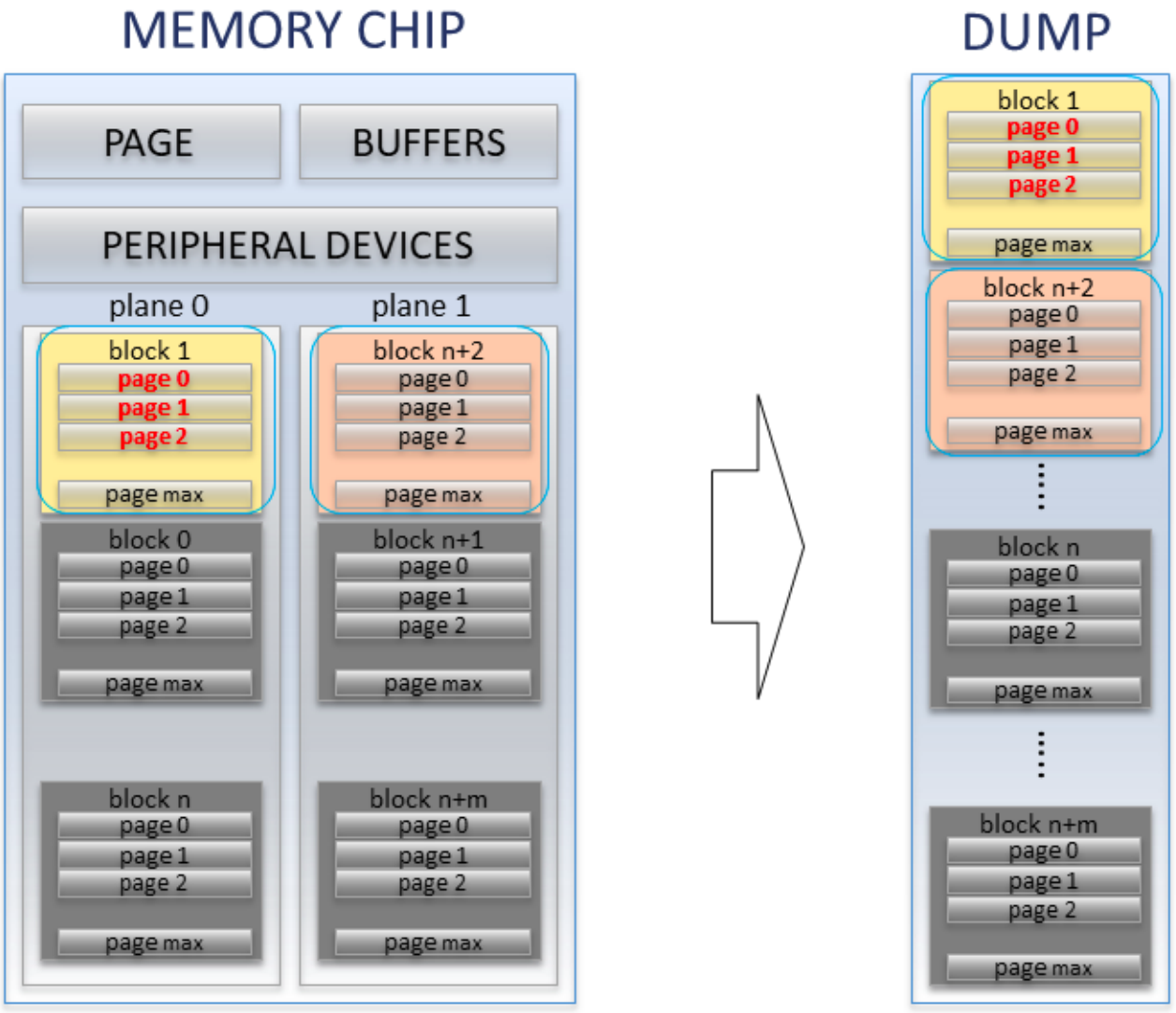
To speed up input/output operations controllers may use parallel allocation modes in which some chunk of data is being allocated between Planes or in

case of Multi-chip (Multi-Crystal) devices between chips. SM controllers may use several allocation modes. Two basic allocation modes that we can find in the majority of NAND controllers and two exceptional ones which are rather rare and likely individual for these controllers.

To reverse engineer and figure out page allocation mode, analysis of bitmap is required or in case of complex multi-chip (Multi-Crystal) device, additional operations.

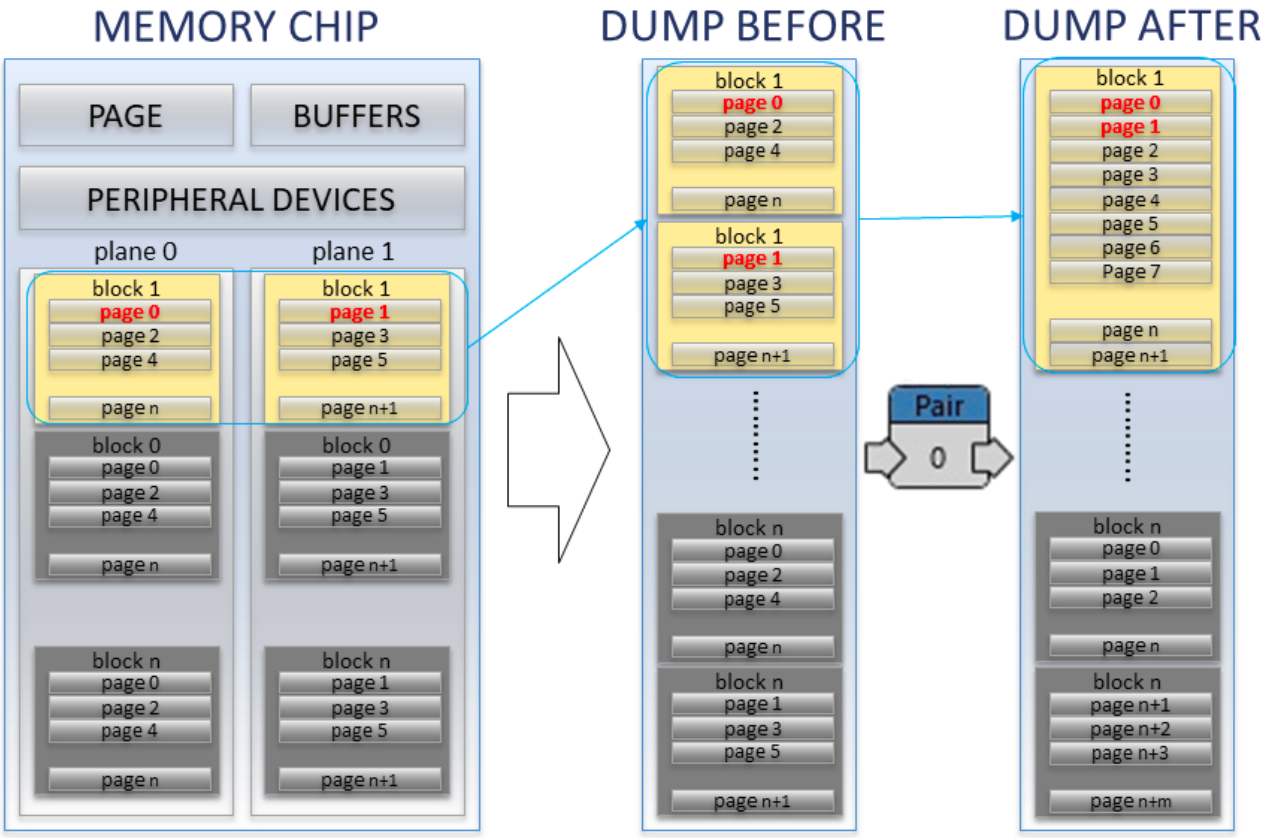
Single Plane Page allocation

This is the base model of page allocation. In this mode, the controller allocates physical pages sequentially to physical blocks of each plane. In these cases, any additional element doesn't have to be used, since data is already sequentially allocated by the controller.



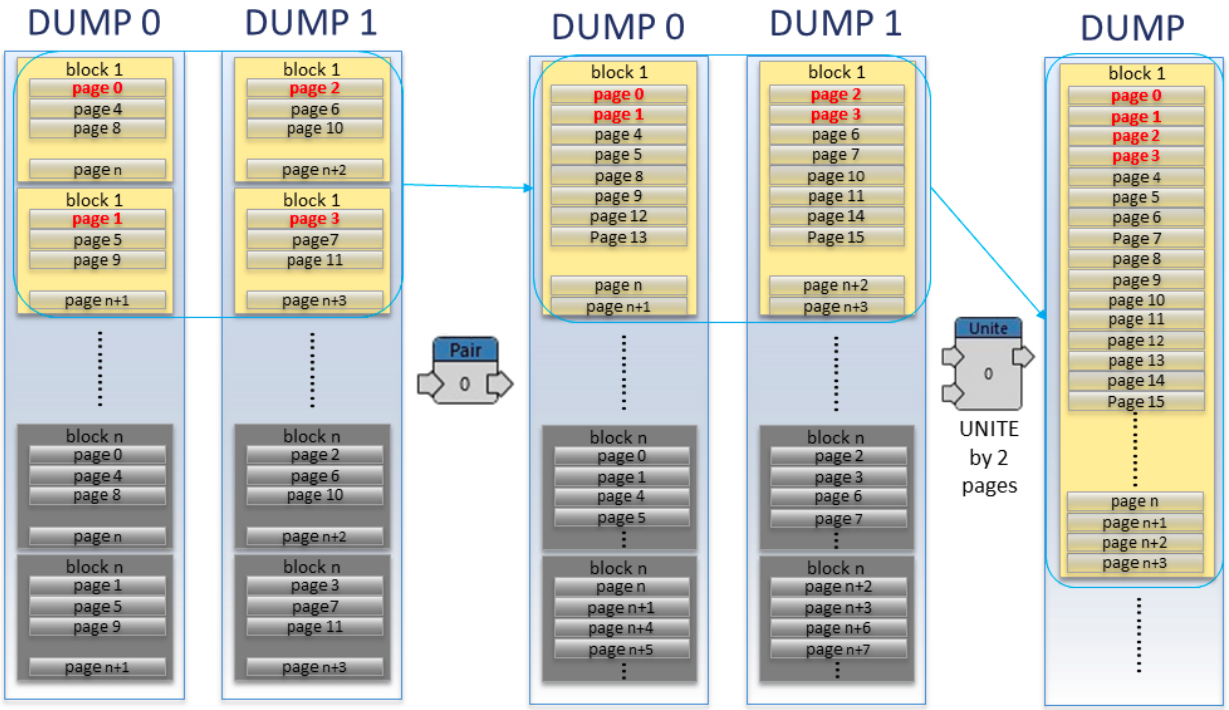
Multi-Plane Page allocation

Parallel allocation of physical pages is the most popular mode used by the controllers to speed up I/O operations. While using this mode controller is allocating physical pages parallelly to the planes of NAND memory. This mode is used by the majority of NAND controllers.



Multi-Plane Page allocation on the chip level

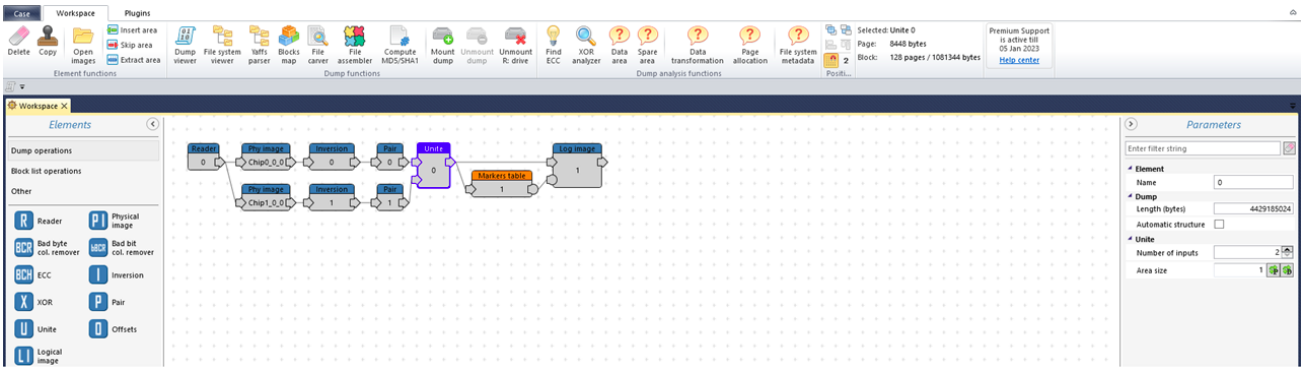
Devices that consist of more than one Chip or Single chips that have more than one Crystal, may allocate data additionally between the Chips/Crystals and their Planes.



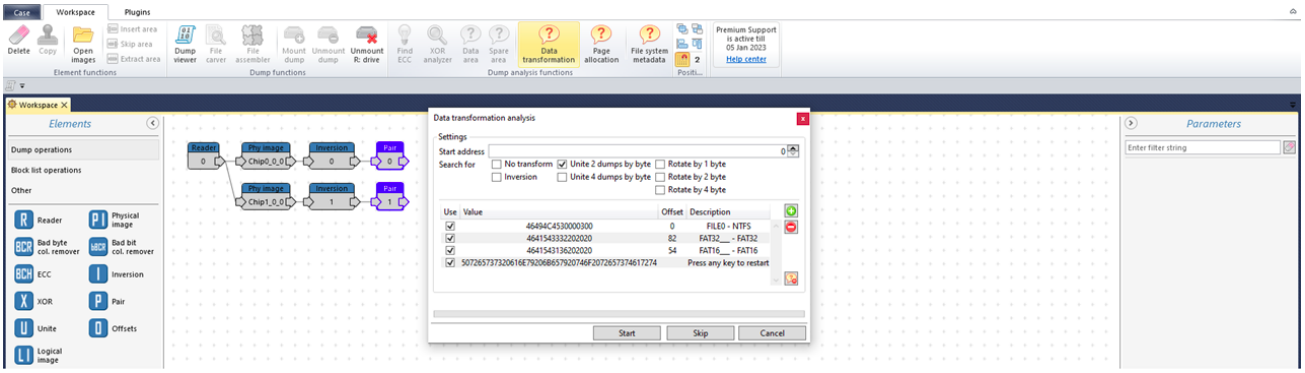
ⓘ For more information about Multi-Plane Page allocation analysis please check this article: Multi-Plane Page Allocation.

1-Byte allocation

Former Silicon Motion controllers which were designed for small capacity SLC chips have used 1 Byte allocation mode. In this mode, the controller was allocating parallelly 1-Byte, but only between memory chips. Allocation between planes was performed normally in physical pages. To join dumps back together is necessary to set up Unite - Area size to 1 Byte, like in the example below.



This allocation mode can be detected using the Data Transformation function. It's necessary to mark dumps, run the tool and set correct option.



8-Byte allocation

This mode is used by SM2236 controllers which are usually placed in high-speed CF cards. In these cases, the controller allocates 8-Bytes parallelly between memory chips. Recovery for these cases is always complex due to the not divisible size of the ECC area by 8 Bytes and in some cases SLC modes.

Block translation in Markers table

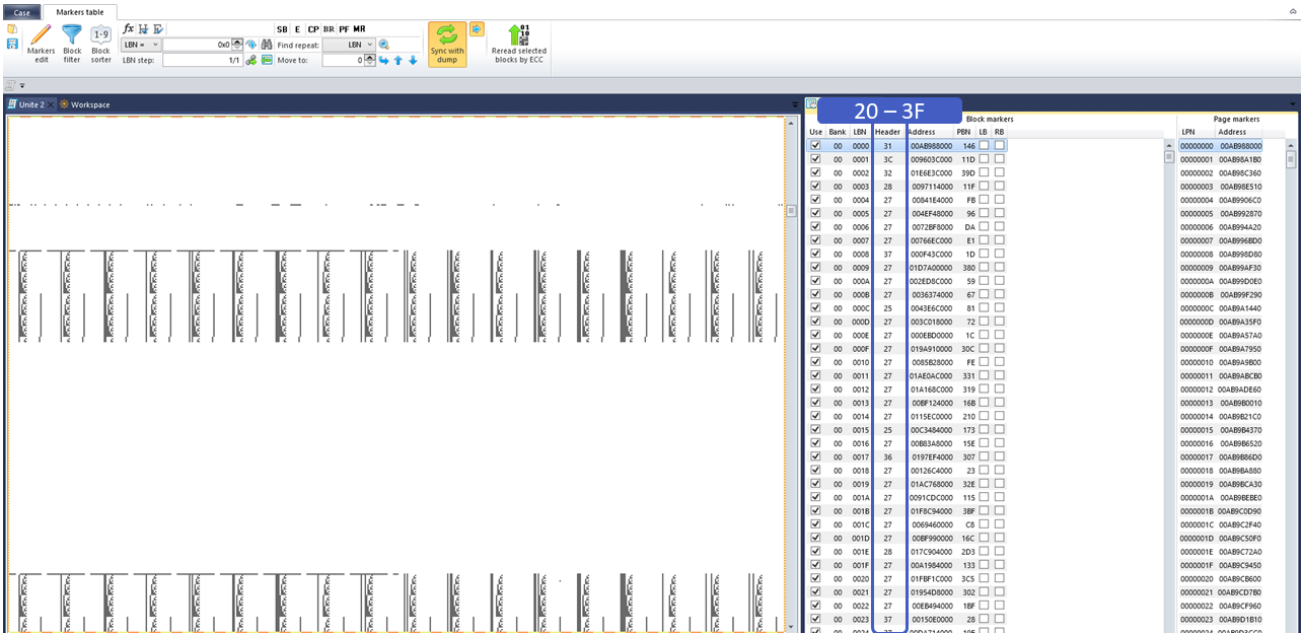
In the last step of each case, it's necessary to arrange logical blocks in the Markers table, and in the case of these controllers, this step is quite simple and looks similar for every model of these controllers.

Structures of Service area

SiliconMotion controllers have exactly the same structure of the Service area.

Header (1 Byte) - The very first byte of the Service area. Defines the type of content and version of data inside the block.

Header values that indicate blocks with data are in the range: [20 - 3F], where a higher value indicates a block with the updated data (in the case when LBNs in both blocks are the same).

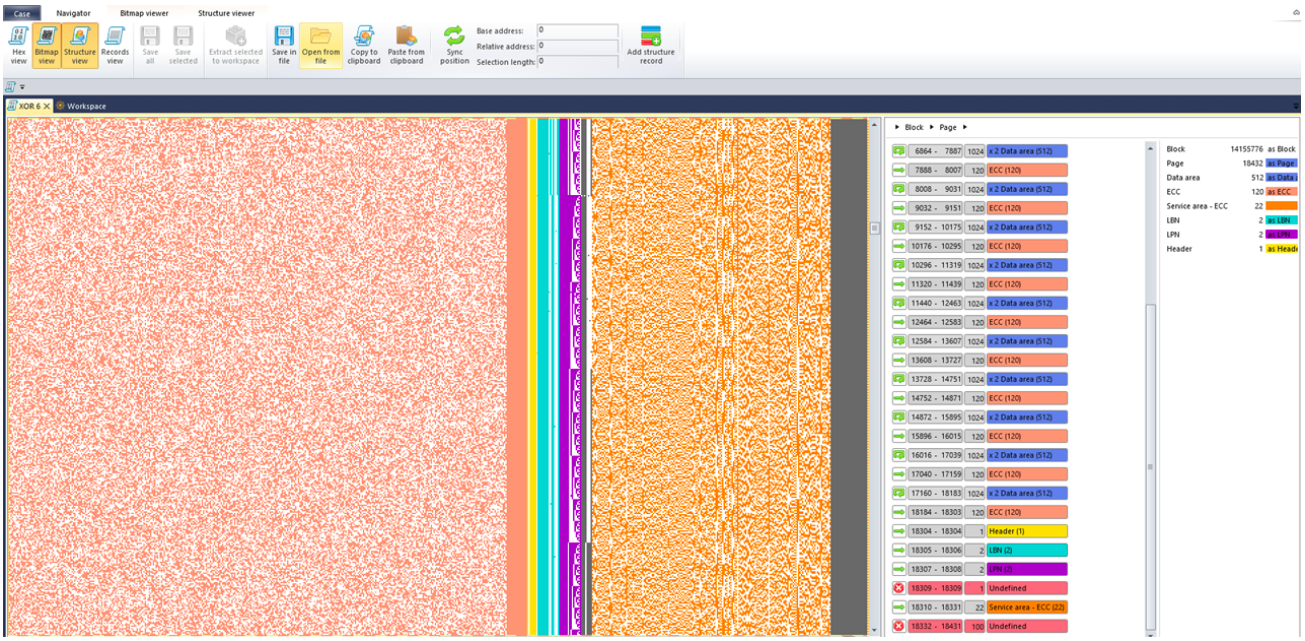


LBN (2 Bytes) - Logical block number is always placed straight after the Header and it has 2 Bytes.

The range of values for LBN will depend on the controller model and capacity, but for most of the cases with these controllers, we can distinguish the following ranges: [0000 - 03FF], [0000-07FF], [0000-0FFF].

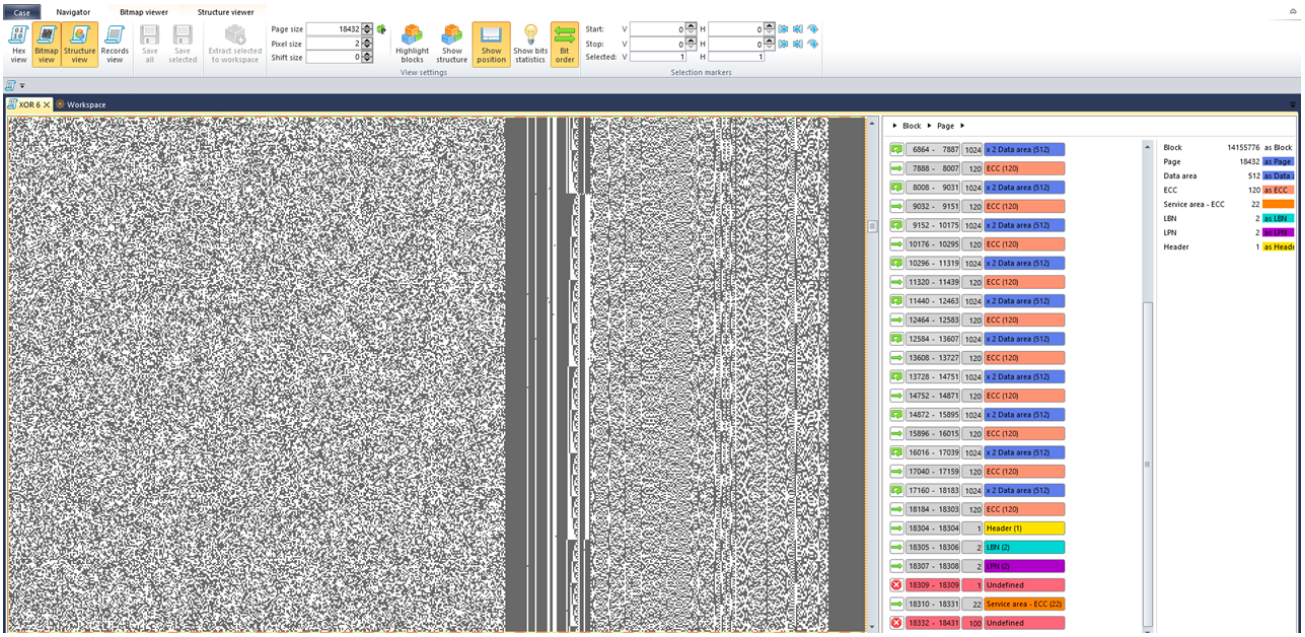
LPN (1-2 Bytes) (Not always available) - Depending on a case LPN may have either 1 or 2 bytes and it's located after LBN.

Bank Number(1 Byte) (Very rarely available) - Defines to which Bank a particular block belongs to. This structure has 1 Byte and it's placed after the LPN or LBN, in case LPN isn't available.



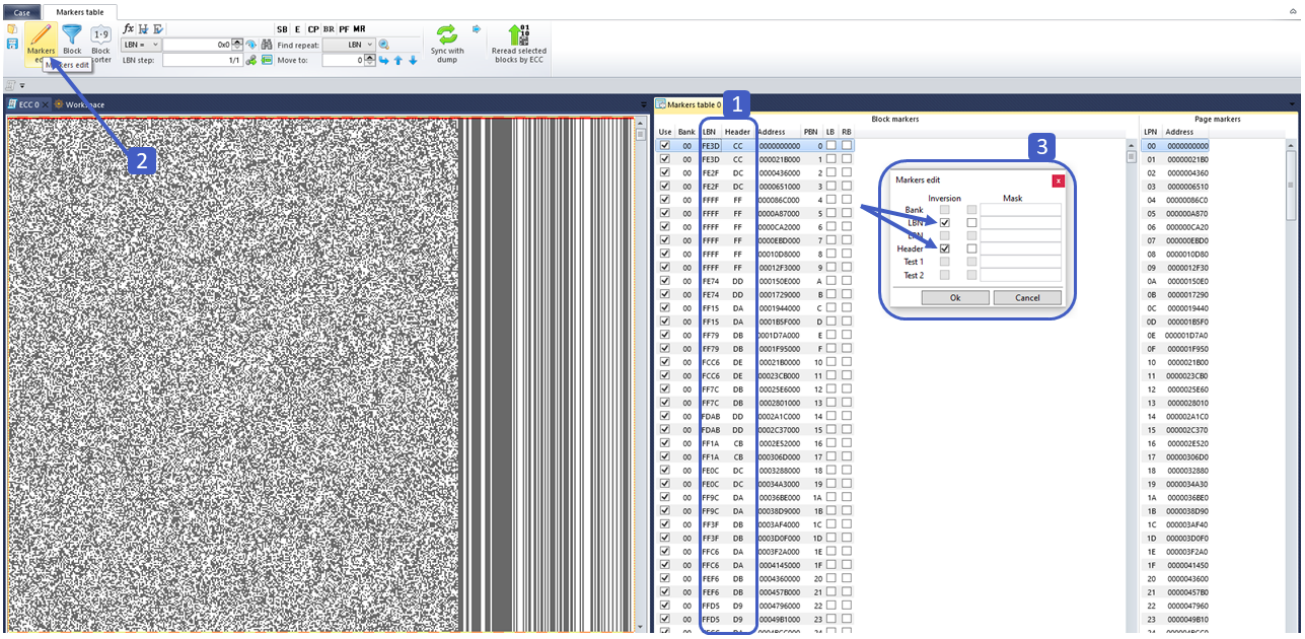
Inverted Service area

The service area in SiliconMotion controllers is almost always Inverted. In such cases, you will be able to see that this area consists of bit 1(grey pixels) mostly.



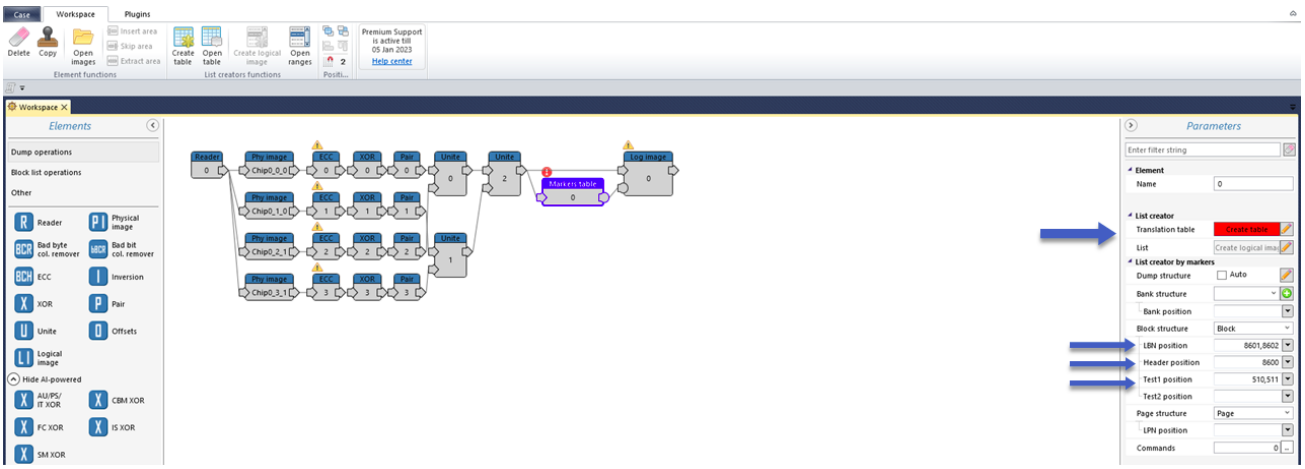
ⓘ SM controllers don't invert particular structures in SA. The controller is Inverting the whole area, so every structure inside.

Inversion can be recognized also after loading values to the Markers table. You will notice that all blocks with data have very high values for LBN and Header. In this situation, it's necessary to Invert the Service area back. To do that open the Markers Edit menu and set up Inversion for all Structures of the Service area.



Analysis of block sequence in Markers table

At first, it's necessary to set positions of LBN and Header in Markers table parameters. Additionally, as a test position, we can set bytes [510,511]. Later 55AA signature on these test positions may help to distinguish the very first block with the file system structure. Afterward to read values from all the blocks in the dump click the "Create table" button.

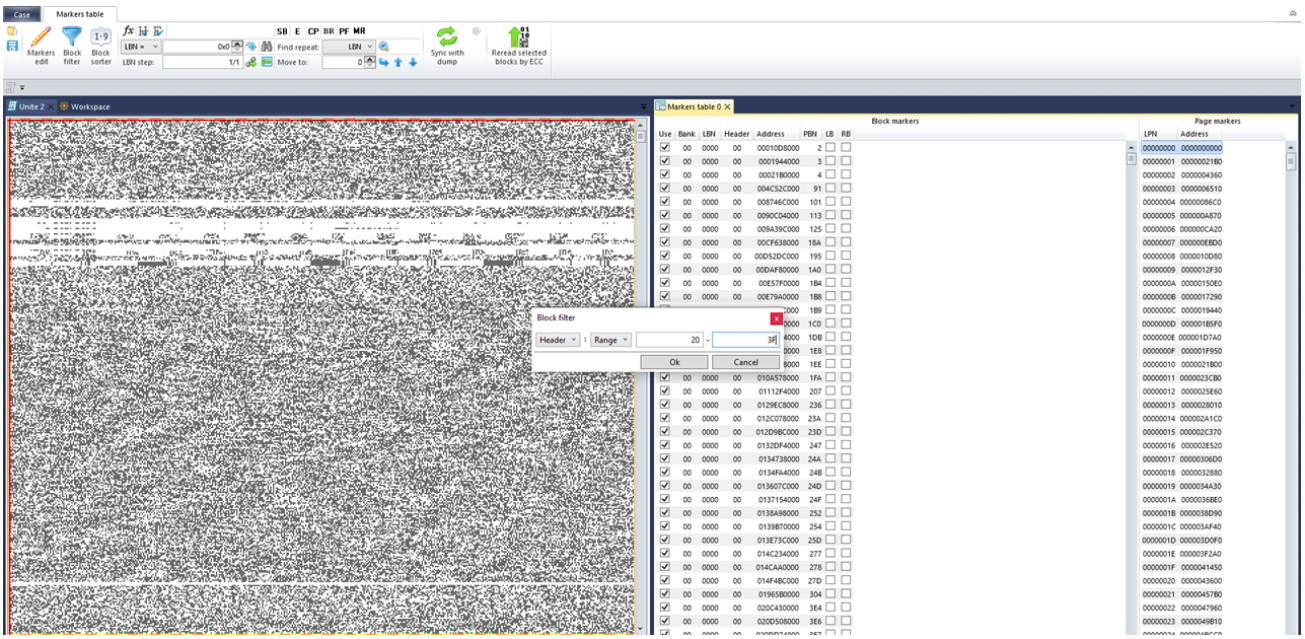


When values are loaded to the Markers table it's necessary to briefly sort blocks and check if it's required to apply Inversion.

> ⓘ Inversion supposed to be applied only on the Structures from the Service area, so LBN and Header. Test 510,511 is located in the Data area and doesn't require to be Inverted.

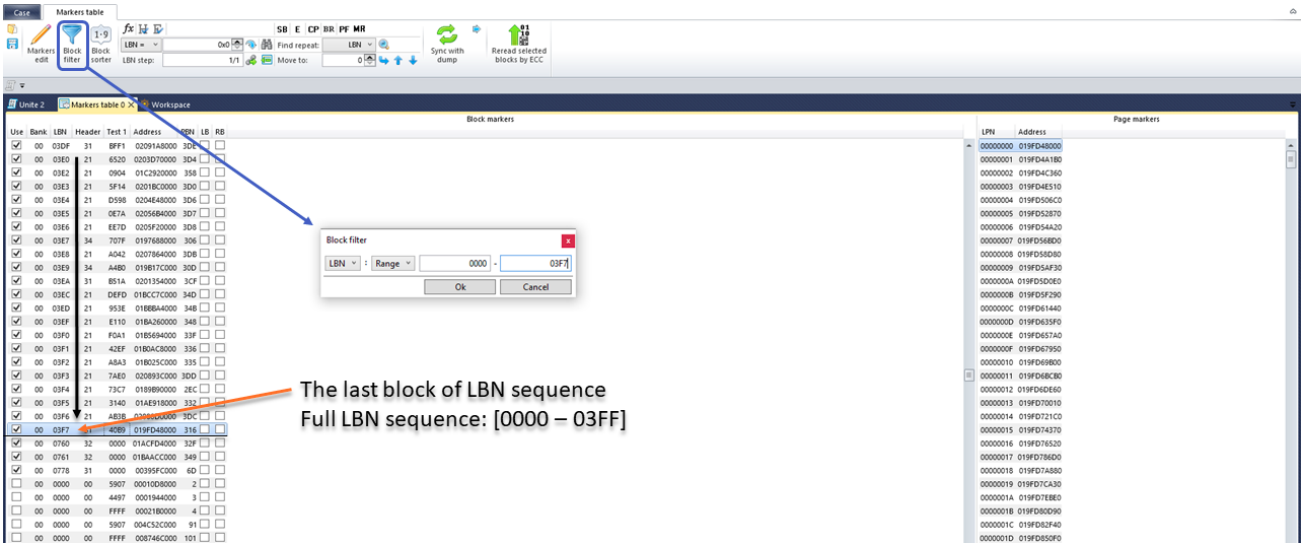
Next, it's necessary to filter blocks by **Header**: Range: [20 - 3F] and once again

sort blocks in MT.



Blocks with data in SiliconMotion cases are starting always from LBN [0000] and in case you have many blocks with this LBN to locate the proper first block, try to find the block with Test_1 equal to 55AAh. This value indicates that the block contains the boot sector and it has a file system structure inside.

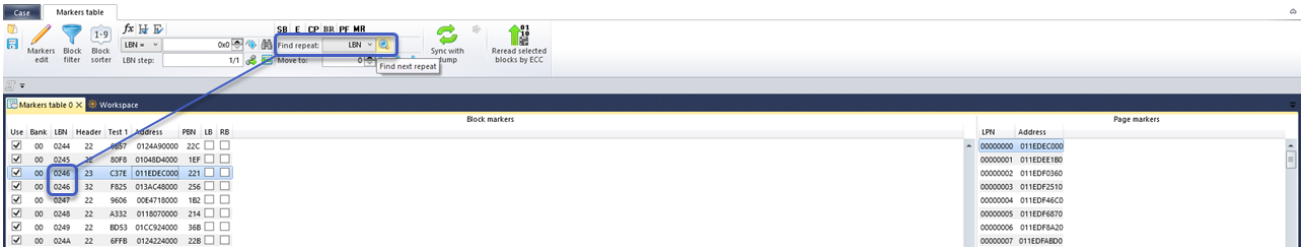
Depending on a chip capacity and amount of recorded data, the LBN sequence will end on different numbers. Examples of full LBN sequences for SM controllers were listed above in the topic Structures of Service area. The last block can be also found manually. You will have to find the LBN number after which the sequence is broken, like in the example below. Then blocks outside the LBN sequence can be filtered out using the Block filter.



LBN filtering, removal of duplicated blocks, and fill gaps function

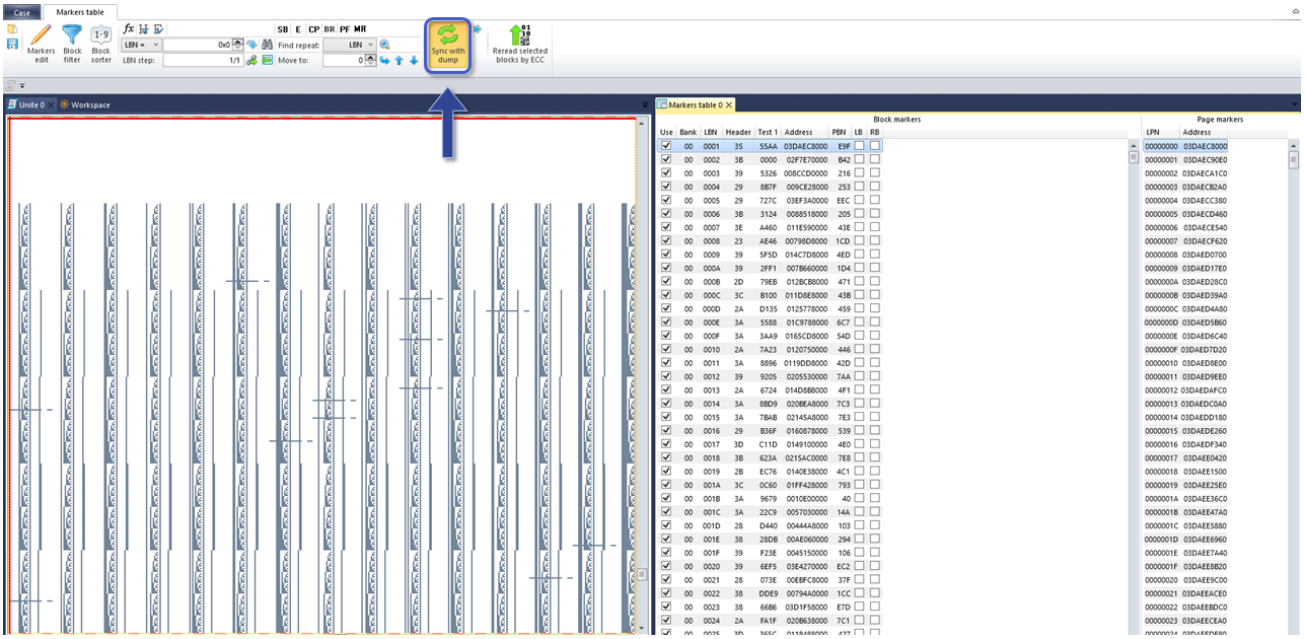
When the LBN sequence has been found, in the next step it's necessary to filter the selected blocks, so remove duplicates and insert missing blocks if there are gaps between LBNs.

Blocks with the same LBNs can be easily found with the find repeat button. After you will set the LBN structure and click the button you will be moved to a position with duplicates. It will be necessary to uncheck repeated blocks and choose only one of them.



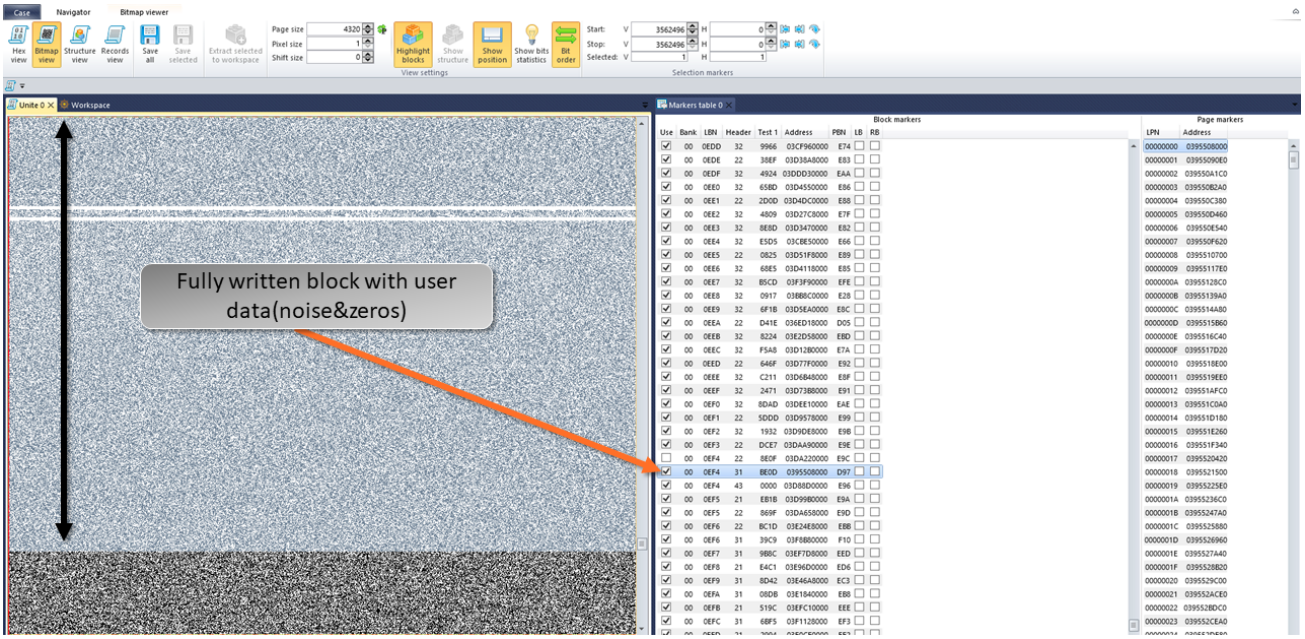
To distinguish which block to choose in case of duplicates it's necessary to compare content and their Headers. Header in case of SM controller indicates also about version of the data, Higher Header number point that block was written later by the controller. Content has to be compared since some of the blocks may not be fully filled with data and these shall be skipped.

Content of blocks can be compared by opening the Markers table and the previous (source) element in side-by-side mode. After that, it's necessary to synchronize MT with the element and turn on the Highlight blocks function to see the exact borders of the block.



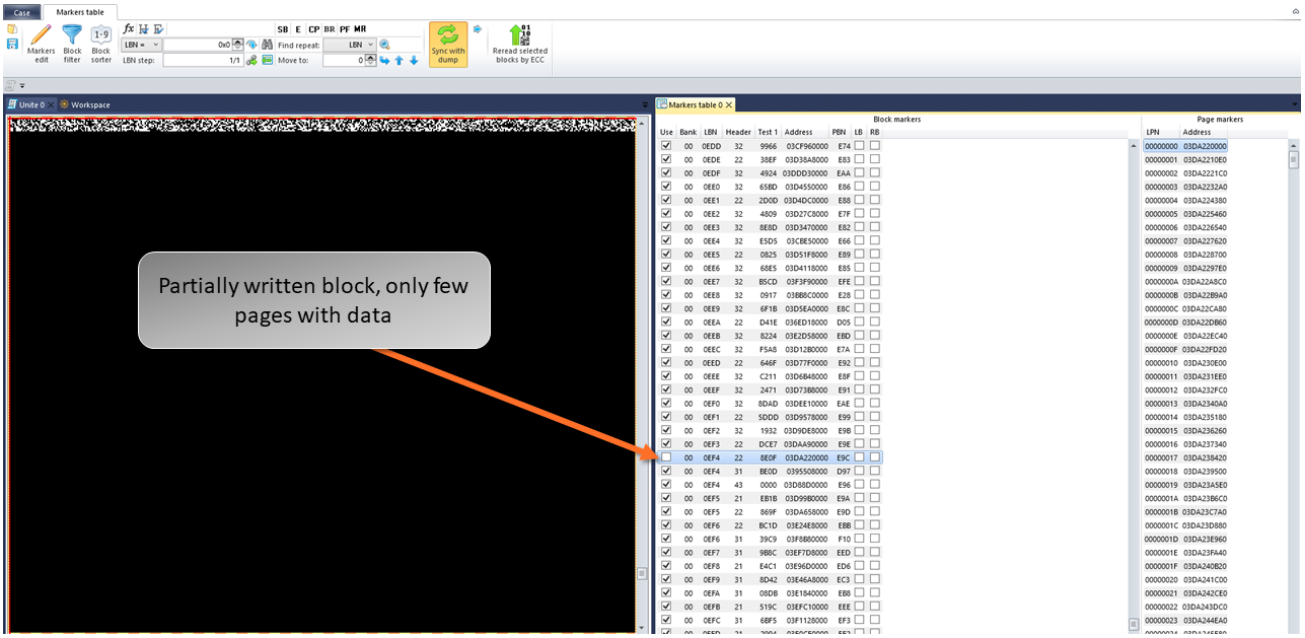
We can distinguish three types of blocks during the comparison of their content.

Example 1. Every page of the block is written by the controller. Blocks with user data mostly contain noise and/or zeros.

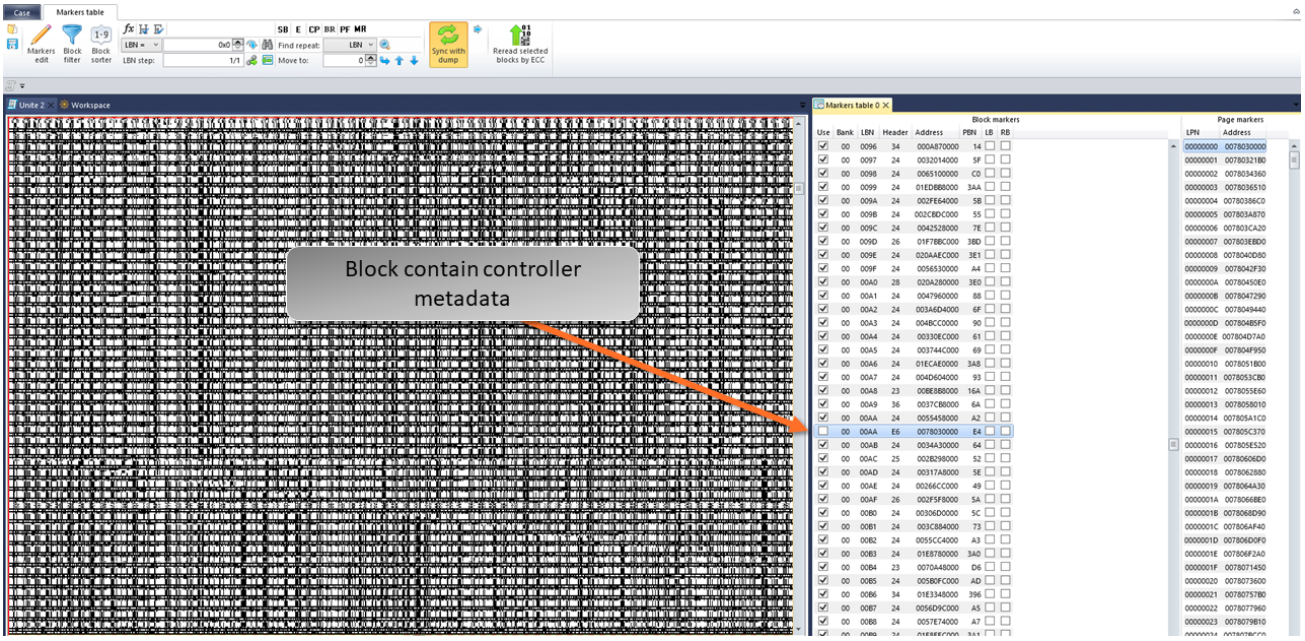


In case you have a lot of fully written blocks with the same LBN it's possible to make a few versions of the Markers table and test each variant of the Logical image.

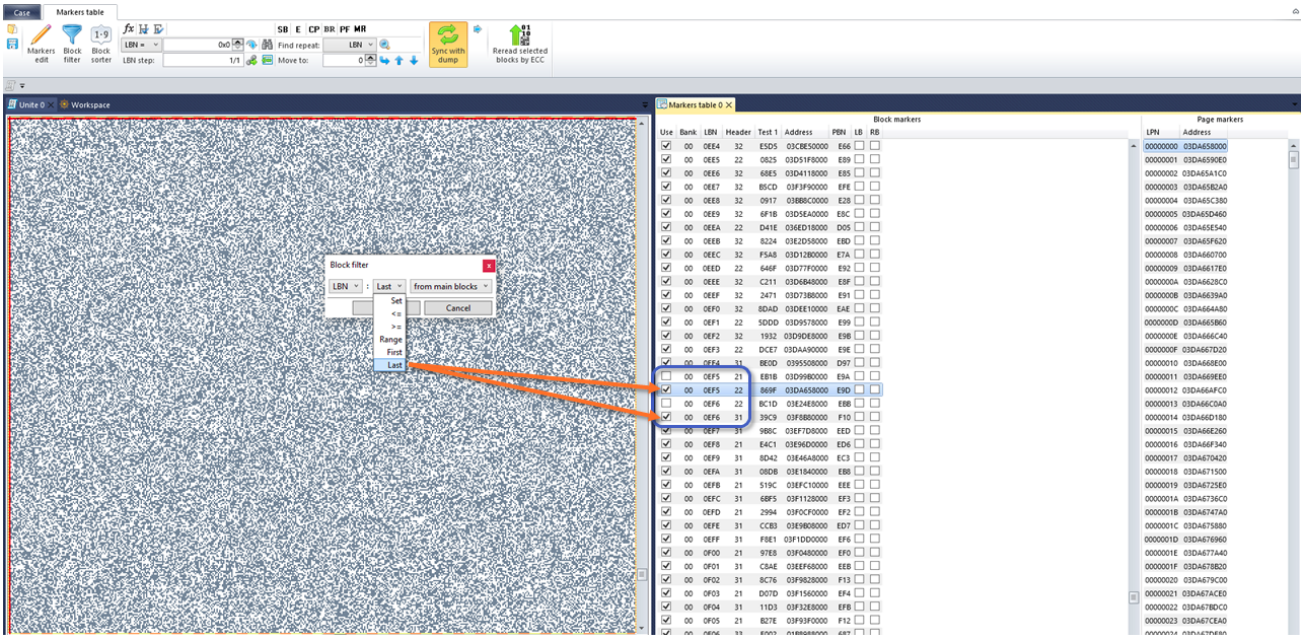
Example 2. Block partially written by the controller. Should be skipped.



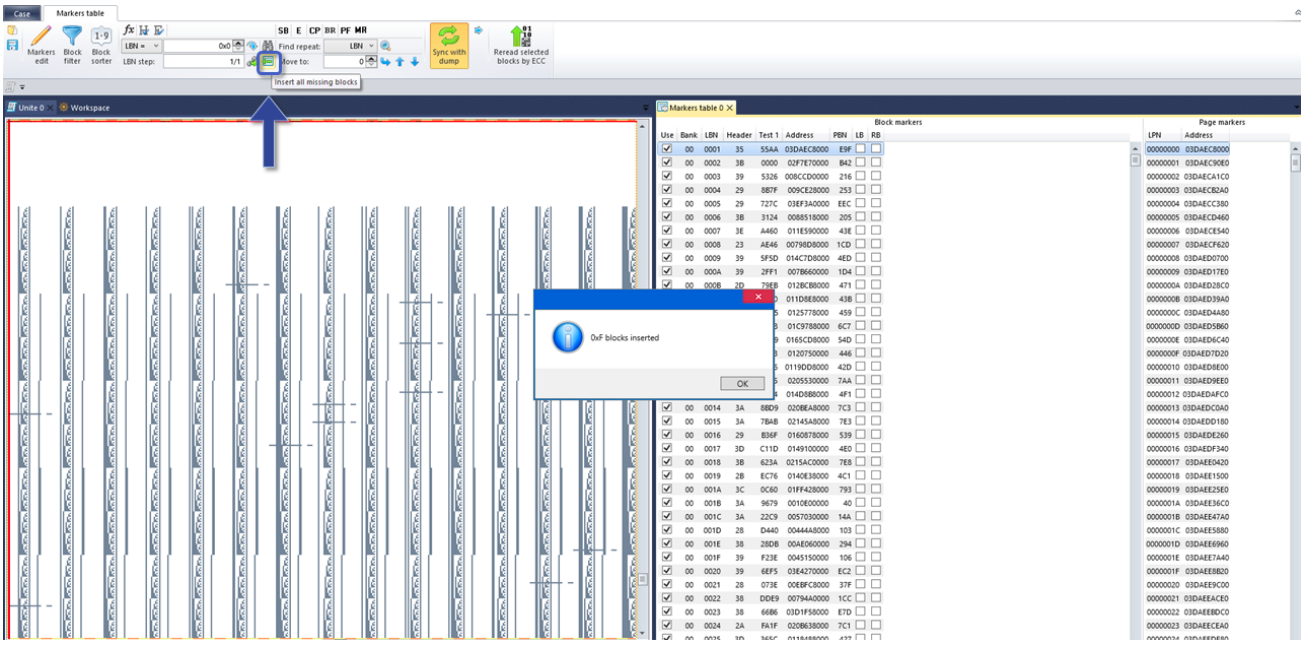
Example 3. Blocks that contain cache data or controller metadata. Such blocks should be skipped. These blocks usually contain a lot of ones and empty pages.



Duplicated blocks can be removed also automatically by choosing the options Last or First in Block filter. This function will select either the Last or the First block from a group of blocks with the same LBN.

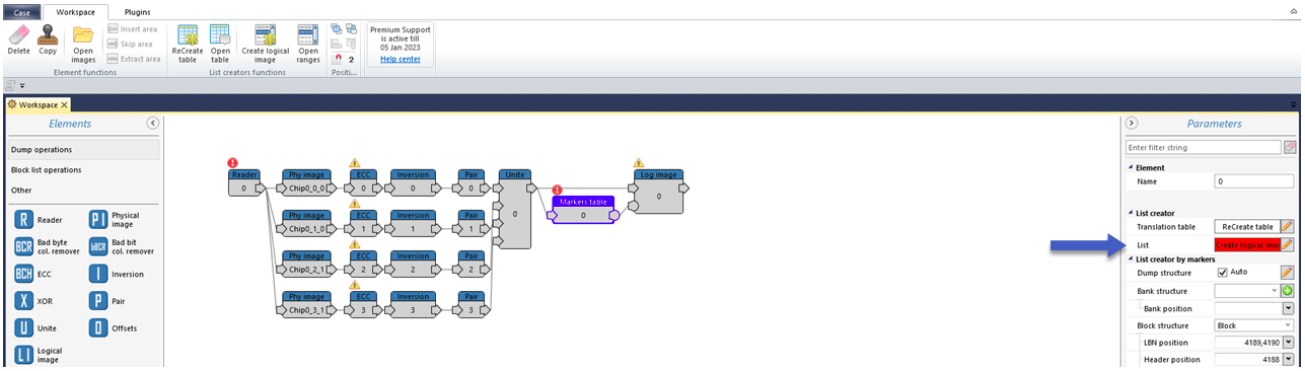


When duplicated blocks have been removed the last step it's necessary to fill the gaps between the blocks to rebuild the file system.



ⓘ This function shall be used only for the cases where file system structures are available. Raw scanning using File Assembler or File carver should be run on the image without "Fill gaps", otherwise some files may be damaged.

When all these steps are complete, click Create Logical Image on the Markers Table parameters menu, connect the logical image element, and check the result.



https://support.rusolut.com/portal/en/kb/articles/silicon-motion-data-recovery